Lehrstuhl für Medieninformatik
Institut für Information und Medien, Sprache und Kultur
Universität Regensburg

Probeklausur

Einführung in die objektorientierte Programmierung mit Java WS1415 - Alexander Bazo 4. Januar 2016

Diese Probeklausur soll Ihnen als Hilfestellung zur Klausurvorbereitung dienen. Die Lösungen zu den Aufgaben werden wir in der Vorlesung besprechen. Viele Aufgabenstellungen aus vergangenen Klausuren finden Sie auf der Seite der SIM (http://www.uni-regensburg.de/universitaet/sim/klausuren/index.html).

Nutzen Sie die Möglichkeit zur *Simulation* der Klausursituation:

- Bearbeiten Sie die Aufgabenstellung nur innerhalb der vorgegebenen 60 Minuten und nur in der Übungssitzung.
- Beantworten Sie die Fragen auf dem Angabenblatt und nutzen Sie gegebenenfalls auch die Rückseiten.
- Nutzen Sie keine Hilfsmittel (Smartphone, Laptop, Bücher, Folien, ...). In der Klausur ist nur ein Stift (kein Bleistift) erlaubt. Sollten Sie ein Wörterbuch benutzen wollen/müssen, sprechen Sie dies **vor** der Klausur mit dem Dozenten ab.
- Täuschungsversuche werden - in der richtigen Klausur - durch entsprechende Maßnahmen bestraft.
- Bei Fragen oder Unklarheiten zur Aufgabenstellung wenden Sie sich an die Aufsichtspersonen/Tutoren.
- Beantworten Sie die Fragen so präzise und kurz wie möglich und geben Sie keine mehrdeutigen (oder mehrere) Lösungen ab.

Zur Beantwortung der Programmieraufgaben ist die Sprache Java zu verwenden. Gehen Sie davon aus, dass alle Aufgaben im Rahmen der bekannten *GraphicsApp*-Umgebung gelöst werden. Wählen Sie sinnvolle Bezeichner und Namen und achten Sie auf korrekte Sichtbarkeit und Syntax. Sie haben 60 Minuten zur Bearbeitung der Klausur Zeit und können 60 Punkte erreichen. Die Punkteangaben bei den Aufgaben dienen auch dazu, Ihnen bei der Zeiteinteilung zu helfen.

Achtung: Die eigentliche Klausur wird ähnlich aufgebaut sein. Der Umfang der einzelnen Blöcke und die Form der Fragen in dieser Probleklausur sind allerdings kein Gewähr für die endgültige Form.

Theorie und Codequalität

1 Vererbung (3 Punkte)

Beschreiben Sie kurz die beiden Begriffe **Subklasse** (Kindklasse) und **Interface**.

2 Codierung und Qualität (5 Punkte)

Beschreiben Sie kurz die beiden Begriffe **Decomposition** und **Top-Down** und erläutern Sie deren Zusammenhang.

Nennen Sie **zwei** Aspekte guter Codequalität und erläutern Sie kurz, warum diese für die Generierung hochwertigen Programmcodes relevant sind.

3 Datenstrukturen (5 Punkte)

Beschreiben Sie kurz und präzise - an einem Beispiel - einen sinnvollen Verwendungszweck für die `HashMap`-Klasse.

Sie entwickeln ein System zur Verwaltung von Warenbeständen. Warum eignet sich eine `ArrayList` besser zu Erfassung der einzelnen Produkte als ein *Array*?

Code verstehen und verbessern

4 Codeverständnis (5 Punkte)

Folgende Methode sei gegeben:

```
1  private void doStuff(Image image) {
2          int[][] pixels = image.getPixelArray();
3          for(int row = 0; row < pixels.length; row++) {
4                  for(int col = 0; col < pixels[0].length; col++)
                       {
5                          Color color =  new Color(pixels[row][
                              col]);
6                          int value = (color.getRed() + color.
                              getGreen() + color.getBlue()) / 3;
7                          color = new Color(value, value, value);
8                          pixels[row][col] = color.toInt();
9                  }
10         }
11         image.setPixelArray(pixels);
12 }
```

Beschreiben Sie kurz und präzise, was die Methode konkret macht. Gehen Sie davon aus, das der gegebene Code fehlerfrei ist.

5 Funktionen, Parameter und Variablen (7 Punkte)

Gegeben sei folgender Code:

```
1  public class ExampleClass {
2          private static final int VALUE_A = 42;
3          private int value_b = 1337;
4
5          public ExampleClass(int value_b) {
6                  println(VALUE_A);
7                  println(this.value_b);
8                  this.value_b = value_b;
9          }
10
11         public void functionA() {
12                 int value_b = VALUE_A;
13                 println(VALUE_A);
14                 println(value_b);
15                 functionB(this.value_b);
16                 println(VALUE_A);
17                 println(value_b);
18         }
19
20         private static void functionB(int value_b) {
21                 value_b = VALUE_A + value_b;
22                 println(value_b);
23         }
24
25         public static void run() {
26                 ExampleClass exampleClass = new ExampleClass
                       (-42);
27                 exampleClass.functionA();
28         }
29 }
```

Welche Ausgabe wird beim Aufruf der Methode `run` generiert? Gehen Sie davon aus, das der gegebene Code fehlerfrei ist.

6 Fehler finden (5 Punkte)

Finden und beschreiben Sie kurz die Bugs, syntaktischen Fehler und Konventionsverstöße in dem folgenden Code-Beispiel. Markieren Sie die fehlerhafte Stellen im Code und notieren Sie Ihre Antworten jeweils unter Angabe der Zeilennummer mit einer kurzen Beschreibung des Fehlers bzw. des Verstoßes.

```
1
2  import java.util.ArrayList;
3
4  public class ErrorClass {
5          public static final int aFixedValue = 42;
6          private ArrayList<String> someStrings;
7
8          public ErrorClass(String aString) {
9                  someStrings.add(aString);
10         }
11
12         public void doSomething() {
13                 int[] numbers = new int[10];
14                 for(int i = 0; i <= numbers.length; i++) {
15                         numbers(i) = aFixedValue;
16                 }
17
18                 println("done")
19         }
20 }
```

Code implementieren

7 Studenten, Dozenten und die Mensa (14 Punkte)

Modellieren Sie eine Reihen von Klassen, die das Bezahlen in einer Mensa abbilden. Kunden (*Customer*) können Speisen (*Food*) kaufen, wenn Sie über ausreichend Geld (*Balance*) verfügen. Für jede Speise gibt es einen regulären sowie zwei reduzierte Preise für Dozenten (*Lecturer*) und Studenten (*Student*). Bauen Sie Ihre Lösung auf der gegebenen Klasse `Person` sowie dem Interface `Customer` auf.

```
1  public class Person {
2          protected String name;
3          protected String id;
4
5          public Person(String name, String id) {
6                  this.name = name;
7                  this.id = id;
8          }
9
10         public String getName() {
11                 return name;
12         }
13
14         public String getId() {
15                 return id;
16         }
17 }
```

```
1
2  public interface Customer {
3          public boolean payFor(Food Food);
4  }
```

Lösen Sie folgenden Aufgabenstellung:

Implementieren Sie die abstrakte Klasse `Food`. Diese verfügt über drei - ebenfalls abstrakte - Methoden `getRegularPrice`, `getLecturerPrice` und `getStudentPrice` die jeweils einen Wert vom Typen *double* zurückgeben und keine Parameter benötigen.

Entwerfen Sie danach die Klasse `MensaCustomer`, die von `Person` erbt und das Interface `Customer` implementiert. Die Klasse verfügen über eine Instanzvariable vom Typ `double` in der das Guthaben der jeweiligen Person gespeichert wird. Im Konstruktor der Klasse wird ein initiales Guthaben als Parameter übergeben und gesetzt. Implementieren Sie das Bezahlverfahren in der vererbbaren Methode `pay(double cost)`.

Dabei gilt: Verfügt die Person über genügen Geld um den - als Parameter übergebenen - Preis zu bezahlen, wird das Guthaben reduziert. Die Methode gibt den Wahrheitswert ***true*** zurück. Im anderen Fall wird nur ***false*** zurückgegeben. Diese Methode wird in der, vom Interface vorgeschriebenen Methode `payFor` mit dem regulären Preis (aus `Food`) aufgerufen.

Legen Sie danach die Klassen *Student* und *Lecturer* an. Beide Klassen erben von `MensaCustomer`. Implementieren Sie für beide neuen Klassen das korrekte Bezahlverfahren durch sinnvolles Nutzen der geerbten Methoden.

Implementieren Sie hier die notwendigen Klassen (`Food`, `MensaCustomer`, `Student` und `Lecturer`):

8 Kreise und Überschneidung (8 Punkte)

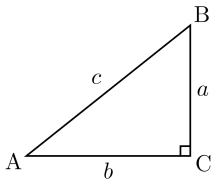
Schreiben Sie eine öffentliche Funktion `doCollide` die zwei Parameter vom Typ `Circle` übergeben bekommt und feststellt, ob sich diese Kreise überschneiden. Das Ergebnis wird als Wahrheitswert zurück gegeben. Für die Implementierung müssen Sie die vorgegeben Klasse `Circle` verwenden. Sie müssen nur die Funktion schreiben und benötigen keine umschließende Klasse.

```
1  public class Circle {
2          private double xPos;
3          private double yPos;
4          private double radius;
5
6          public Circle(double centerX, double centerY, double
               radius) {
7                  this.xPos = centerX;
8                  this.yPos = centerY;
9                  this.radius = radius;
10         }
11         public double getX() {
12                 return xPos;
13         }
14         public double getY() {
15                 return yPos;
16         }
17         public double getRadius() {
18                 return radius;
19         }
20 }
```

Hinweis: Für die Berechnung der Distanz zwischen zwei Punkten innerhalb des Koordinatensystems können Sie den *Satz des Pythagoras* verwenden. Dieser Satz beschreibt die Seitenverhältnisse innerhalb eines rechtwinkligen Dreiecks:

$$a^2 + b^2 = c^2.$$

Folgende Grafik verdeutlicht die Zusammenhänge:



Implementieren Sie hier die Lösung zu Aufgabe 8. Sie dürfen die Methoden `sqrt(double a)` und `pow(double a, double b)` der Klasse `Math` aus dem Paket `java.lang` nutzen.

9 Wortähnlichkeit (8 Punkte)

Ergänzen Sie die gegebene Funktion `getWordSimilarity`. Diese berechnet ein Ähnlichkeitsmaß zwischen zwei gegebenen Wörtern in Form eines Prozentwerts (`double`). Es gilt: Die Wörter werden zeichenweise überprüft, d.h. der erste Buchstabe des einen Worts wird mit dem ersten Buchstaben des zweiten Worts überprüft, usw.). Bei unterschiedlicher Länge werden nur so viele Stellen überprüft, wie im **kürzeren** Wort vorhanden sind. Aus den gezählten, übereinstimmenden Stellen wird eine prozentuale Ähnlichkeit bestimmt. Dazu wird die Anzahl der Buchstaben im **längeren** Wort berücksichtigt. Ein paar Beispiele:

- *Haus* und *Maus* stimmen zu **75%** überein.
- *Käsetheke* und *Außenbordmotor* stimmen zu **7%** überein.
- *Schule* und *Universität* stimmen zu **0%** überein.

Hinweis: Ein Zeichen an einer bestimmten Stellen innerhalb eines Strings können Sie mit der Methode `public char getCharAt(int index)` der `String`-Klasse auslesen.

```
public double getWordSimilarity(String word1, String word2) {









}
```